SIMPLE, FAST, AND ROBUST SELF-LOCALIZATION IN ENVIRONMENTS SIMILAR TO THE ROBOCUP ENVIRONMENT [1]

MohammadTaghi Hajiaghayi[a], Mansour Jamzad[b]
[a] *Laboratory for Computer Science, Massachusetts Institute of Technology*
hajiagha@lcs.mit.edu
[b] *Department of Computer Engineering, Sharif University of Technology*
jamzad@sharif.edu

Self-localization is important in almost all mobile robotic tasks. For robot soccer as well, an efficient and fast self-localization method is a necessary prerequisite. Techniques for robot self-localization have been extensively studied in the past, but most of them cannot be used alone in the RoboCup environment. Often it is necessary to combine several methods to get better performance. In this paper, we present a fast and robust algorithm for self-localization implemented in the SHARIF-CE *robotic soccer team for RoboCup 2000. This algorithm is based on* lines *detection and has two different approaches for self-localization using lines. We also present some experimental results.*

1 INTRODUCTION

Self-localization has an important role in mobile robot tasks, especially in RoboCup. Dribbling, passing and replacing of robots all need a fast and accurate self-localization.

Self-localization can be considered in two aspects: *local* and *global*. We need both of them in RoboCup: in the former aspect, we estimate the distance of one special object from the robot, such as distances of walls, ball, goals or other soccer robots; but in the latter, we find the position and direction of a robot with respect to a global fixed coordinate frame. Obviously, if we know positions and directions of all objects in the field, then we can find the positions of objects with respect to each other very easily. However, global self-localization usually requires more time, has higher processing costs, etc. In most situations, we can

[1] The research was done when the first author was a member of the robotic soccer project at Sharif University of Technology (SHARIF-CE robotic soccer team).

abandon global self-localization and use only local self-localization to do robot tasks, as we did in the *SHARIF-CE* robotic soccer team. This was one of the key factors in the victory of our team in RoboCup 1999 and RoboCup 2000 [14, 15].

Nevertheless, this does not decrease the importance of global self-localization, because tasks like dribbling , passing and replacing of robots cannot be done well without a robust global self-localization. For this reason, we present in this paper an algorithm from the *SHARIF-CE* robotic soccer project implemented for global self-localization. In this algorithm, we simply used a local self-localization in order to get a global self-localization. It is worth mentioning that there are s many techniques for global self-localization (see [1, 3, 13] for a survey), but most of them need very special environments and none of them by itself can be used in a very dynamic environment such as the RoboCup field. However, we can design better methods using a combination of these techniques. Self-localization in the RoboCup environment and other mobile environments has also been studied in [5, 6, 11].

To solve the global self-localization problem we can use a wide variety of sensors (e.g., odometer, sonars, vision, compasses, laser range finder, other sensors, or combinations thereof). In this paper, we consider a vision-based algorithm which uses only a single CCD color camera; however, other sensors can also be added and the algorithm can be adapted to use them for more accurate global self-localization. This property enables the algorithm to be easily generalized and used in other mobile robot navigation tasks. For simplicity, in the rest of this paper we use the term *self-localization* instead of *global self-localization*.

The paper is structured as follows. In section 2, we mention the objects used in self-localization named *lines*. Then, in section 3, we briefly discuss methods of finding lines. The main algorithm is presented in section 4. Finally, in section 5, we conclude and sketch future work.

2 WHY LINES?

The objects that we use in our algorithm are *lines*. We define a line as a boundary of two regions with different colors. These colors can be chosen among yellow, blue, green and white, which are the main colors of the RoboCup field. Thus all corner lines, goal lines, and ground lines are included in this definition. The main reason we use lines instead of other objects, such as corners or other markers, is that lines are the intersection of two adjacent large regions on the field with different colors. These two properties, i.e., the difference of colors and the largeness of the regions, enable us to detect a line by seeing only some parts of that line. Since robots and the ball continually move in the field, we see only parts of some lines. In addition, we use a CCD camera with a view angle of 75 degrees, mounted in front of the robot. However, since we use only one front

view camera, only parts of some lines will be visible. But, if we use other known landmarks such as corners, then we can rarely see all we need for self-localization. However, lines are influenced little with respect to the aforementioned dynamic environment. This is another reason we use them for self-localization.

It is also worth mentioning that there is much noise in the RoboCup environment, e.g., shadows and scripts on the wall, that decreases the accuracy of detection of the other field parameters. However, these factors do not affect the detection of lines by simply adjusting the ranges of the colors.

3 THE METHODS OF LINE DETECTION

It is worth mentioning that each line has two different images, one image in the CCD called the *virtual image* and one in the field called the *original image*. First, we obtain the virtual image and then map it to the field to obtain find the original image. To find the virtual image we must scan the image obtained by CCD camera. This scan can be done by sampling of points from the image instead of scanning the whole of the image. This sample can be chosen simply by Δx and Δy jumps over the pixels from the image. Then we need to find the *boundary points* among these selected points. A boundary point is a point which has two different colors in its neighborhood. After that, we have to find all the boundary lines passing through these points. To do this, first we partition these points into several classes, such that all points of each class have two different colors in their neighborhood and each class is distinguished from others by these two colors. Since in a RoboCup soccer field the mail colors are green(the filed), white(walls and border lines), yellow and blue (goals), thus, we have at most five classes with distinguished colors: green and white, green and yellow, green and blue, blue and white, and, finally, yellow and white. We note that one point can appear in more than one class because it is possible that more than two colors appear in the neighborhood of that point. We have implemented this method and observed interesting results: for an image with 768*586 resolution and $\Delta x = 20$ and $\Delta y = 20$ the number of boundary points was less than 400 points and each class had approximately 100 points on average.

We then found the lines passing through the points of each class. There are some standard methods for line detection [4, 7]; however, we have used the *Hough Transform* method [2, 10] which seems more efficient and more accurate in comparison to other methods.

In Hough transform we assume the equation of a line passing through a point (x_i, y_i) in x, y plain is expressed in the general form of $y_i = ax_i + b_i$. Writing this equation as $b = -x_i a + y_i$, we can draw this line in a, b plain (that is usually called the *parameter space*). For any point (x_i, y_i), we draw its corresponding line in a, b plain. The interesting property of this parameter space is that for all

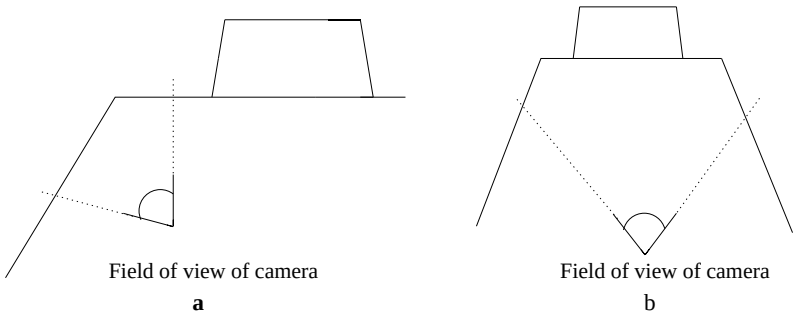

Figure 1: Lines that the camera sees

points that almost lie on a straight line, their corresponding line in parameter space will cross each other in one point. Therefore, by finding the coordinate of this point in parameter space, we have the a, b corresponding to a line equation $y = ax + b$. This line is the solution we are looking for. It passes through a set of edge points (x_i, y_i), $i = 1, 2, 3, ...$ [4].

However, We have changed this algorithm somewhat to obtain more accuracy. For example, we have used a *weight* for each point. The weight of a point is the probability that a point will be a boundary point with respect to the noise of the environment. This probability can be computed in different ways such as counting the points of each color in the neighborhood of the point. The weight of each point can be exercised in Hough Transform or any other line detection algorithm simply by duplication of that point with respect to its weight. In other words, we use more copies of those points which have more weight and thus the points will have more effect in the algorithm. The reader is referred to [12] for further details on use of weight. After doing all of the above operations, we have some classes in which each class has some lines passing through its points. It is worth noting that the number of these lines can be zero, one, or greater than one. If the number of points of a class is less than a threshold we cannot pass any lines through the points of that class. Also, if we consider the corner of the field or the boundary of wall and field (i.e., the class distinguished by green and white colors), we can obtain two intersecting lines (Figure 1.a) or even three lines (Figure 1.b) In addition, some other lines can appear as noisy lines. For example, in Figure 2 there are three lines passing through the points of a class, but the line number 2 is a noisy one and it must not be counted as a true line. We can eliminate such lines by allotting the proper weights to the points, using the slope of the detected lines, or the combination of these methods. After using all these methods, it is also possible that some noisy lines will appear as true lines, but this fact, does not have a large effect on our final algorithms.

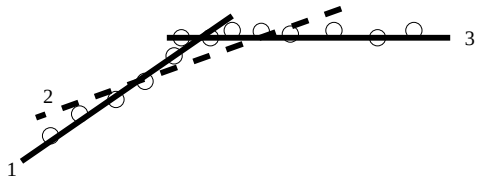

Figure 2: Lines passing through points

After recognizing all virtual images of lines in the image from the CCD camera, we have to obtain the original images (the actual location) of these lines in the field. This operation can be done simply by mapping two points of each line on the virtual image to their corresponding points in the field (to see the mapping method, the reader may consult [8, 9]). However, it is very important to detect lines that are far from the robot with more accuracy, because a little change in the virtual image of a far line can cause a large degradation in the actual location of that line in the mapping process. To gain more accuracy, we may initially find the lines using the Hough Transform method and then use other methods of passing lines through points such as the Total Least Squares method (see [7]) over the points near that line. Using this approach, we detect a line more accurately than the previous method.

Using an efficient implementation, as we have done in our team, we are able to do all the above operations at a rate of 13-18 frames per second while performing the other operations used for object detection and the decision making algorithms that are implemented in real time in a robot.

4 SELF-LOCALIZATION USING LINES

In this section, we discuss further the main points of the algorithm and present two different approaches to the problem of self-localization. But first, we mention some basic definitions.

For self-localization, first we need *global coordinate axes*. A global coordinate axis is placed in front of the blue goal and in the middle of it (see Figure 3). The z-axis is also a perpendicular axis with respect to the x and y axes (the positive direction of z-axis is the up direction). With these definitions each point in the field has an (x, y, z) coordinate. We also assume that the z-coordinate of the robot is zero and thus to locate a robot we need (x, y) coordinates of its center. In addition to the (x, y) coordinates, we also need to know the direction of the robot. This direction is the angle θ between the front vector of a robot and the positive x-axis (see Figure 3).

Hence, the *robot's pose* is the (x, y) center of the robot and its orientation

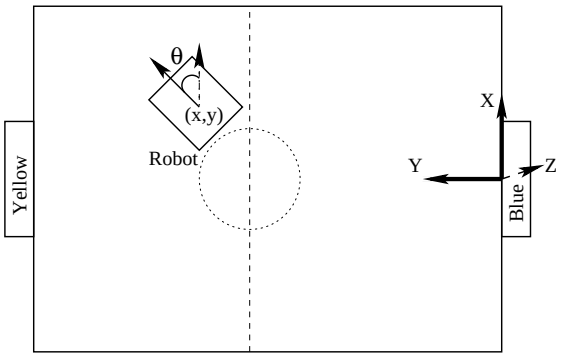

Figure 3: Global coordinate axes and robot's orientation

(the angle θ).

Another important matter concerns the local self-localization of a point P with reference to a line L in the field. We can easily find the distance d and angle θ' with respect to the perpendicular vector L. We use this distance and angle in our algorithm.

We also suppose that we have a history in which we keep the $Histx$, $Histy$, and $Hist\theta$ of our last obtained x, y, and θ of the robot. NHx, NHy, and $NH\theta$ also keep the number of the latest frame in which we have computed the latest x, y, and θ. We note that NHx, NHy, and $NH\theta$ may be different because it is possible that in one frame we are able to compute only x and θ and in the next frame we are able to compute only y and θ.

Using the above definitions and facts, we present two approaches for self-localization. We have implemented both methods, but we used the first method in the final algorithms of the robots.

4.1 First method

Looking at line L from a class C, if we know which of the boundary lines of the field corresponded to L, then we can find θ , x, or y of the robot with respect to L, which is parallel to the y-axis or the x-axis of the global coordinate system. The problem now is that we cannot easily recognize the corresponding field line of L. In a limited region of the robot's field of view, we cannot recognize which of the intersecting lines is parallel to the x-axis and which is parallel to the y-axis (see Figure 1.a). In fact, knowing the class of a line corresponds to knowing the colors of two regions adjacent to this line but we are not being able to completely recognize such lines. To recognize a line completely, we need more information, e.g., if we see one of the goals then the process of line recognition can be done

easily. If we wanted to use many factors to solve this problem, it might cause an explosion of the number of cases, hence the program might become unreliable. In addition, in some cases we cannot recognize the line because of symmetry, e.g., when a robot is very near the wall that it can only see white colors. Hence we need a more general approach. One way is to use factors called *key factors*, e.g., the goals, by which we are able to uniquely recognize a line. Now, the problem is, what do we do if we cannot see any key factors. We assume this line should be one of the boundary lines of the field. Using this assumption, we find the θ, x ,or y of the robot. Now, we refer to our history and see whether these values can be matched to the corresponding values there (we also use NHx, NHy, and $NH\theta$ to check the validity of our history). If they cannot be matched, we use other assumptions and continue to seek the best one. Note that the boundary lines in the field are few, thus this approach can be done efficiently.

The above method has two advantages: first, we prevent the cumulative error in the history by seeing the key factors, and, second, we use our history when appropriate to find an accurate self-localization.

All we have discussed in above relates to one line. But, we usually see more than one line in the image from the CCD camera and thus, by using more than one line, we will be able to find the robot's pose more accurately. Then we can load the best result in the history.

4.2 Second method

There is another interesting method for global self-localization which needs some pre-processing. In this approach, we use a simulator program, given the x, y, and θ of the center of a robot, it shows the objects that the robot sees if we place the robot in the given position (suppose that the field is empty). Now, we run the line detection algorithm on this simulated image and find the classes and the lines of each class precisely (note that in this simulated image there is no noise and hence everything is precise). If we do these operations for some of the points of the field and some orientations (with regard to expected accuracy), we can find a mapping from the robot's pose to the classes and lines of each class. Using a reverse mapping, we can now compare the classes and lines of each class at a point of the field to the pre-processed data and find the best x, y, and θ with regard to the errors . We note that the error of each comparison can be computed using different methods, one being computing the difference between the sum of the squares of the parameters of the lines obtained from the image of the CCD camera (e.g., the slope of a line) and the sum of the squares of the parameters obtained from the simulated program. In this method, we use the history only for breaking the ties between symmetric situations. We must use an efficient method for the reverse mapping. One way to obtain an efficient

reverse mapping is the use of neural networks, such that we use the pre-processed information for learning of correspondence between lines and the robot's pose. Then we will use this function for self-localization in the game.

5 CONCLUSIONS

We have implemented the above algorithms in the *SHARIF-CE* middle size robotic soccer project. We used a Pentium processor and a Genius grabber card for capturing the images and self-localization. Using these very simple instruments, we were able to do the self-localization process in 10-15 frames per second. This approach is very efficient, especially for robots like the goal-keeper that have a private region in which the ball and other robots rarely appear. The only situations in which it would be possible that this algorithm produces incorrect results are the situations in which the robot is surrounded by two or more robots and (in this situation) the robot turns left or right. To solve this problem we may use these solutions:

1. Using more than one camera on the robot body to gain more information from the environment, allows us to find the robot's pose with more accuracy. This solution requires more processing, and therefore a faster processor is needed.

2. The use of other sensors like an odometer or an electronic compass will provide us one or two parameters among x, y, and θ; hence we can map the detected lines to the corresponding lines of the field with more accuracy.

The method presented in this paper can not only be used in the RoboCup environment, but can also be used in other robotic tasks where there is a limited number of colors and the intersections of color regions make many lines. The benefit of this algorithm is that it is not complicated and needs very simple hardware, i.e., a regular camera and a regular processor. Therefore it can be used in situations with limited resources.

6 ACKNOWLEDGMENTS

We would like to thank Hamidreza Chitsaz, Vahab Mirrrokni, Abbas Heydarnoori, Ehsan Chiniforooshan and other SHARIF-CE team members who helped us to implement these algorithms.

References

[1] Borenstein J., Everett H.R., and Feng L. *Navigating Mobile Robots: Systems and Techniques*. A.K. Peters, Ltd., 1996.

[2] Duda R. and Hart P. Use of the hough transformation to detect lines and curves in the pictures. *Comm. of the ACM*, 15(1):217–241, 1972.

[3] Fox D. *Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation*. PhD thesis, Institute of Computer Science III, University of Bonn, Germany, 1998.

[4] Gonzalez R.C. and Woods R.C. *Digital Image Processing*. Addison-Wesley, 1992.

[5] Gotmann J.S., Weigel T., and Nebel B. Fast, accurate, and robust self-localization in the robocup environment. In *RoboCup-99, Lecture Notes in Artificial Intelligence*, volume 1856, pages 304–317. Springer, Berlin, 2000.

[6] Iocchi L. and Nardi D. Self-localization in the robocup environment. In *RoboCup-99, Lecture Notes in Artificial Intelligence*, volume 1856, pages 318–330. Springer, Berlin, 2000.

[7] Jain R., Kasturi R., and Schunk B.G. *Machine vision*. McGraw-Hill, 1995.

[8] Jamzad M., Foroughnassiraei A., Chiniforooshan E., Ghorbani R., Kazemi M., Chitsaz H., Mobasser F., and Sadjad S.B. Middle sized soccer robots: Arvand. In *RoboCup-99, Lecture Notes in Artificial Intelligence*, volume 1856, pages 61–73. Springer, Berlin, 2000.

[9] Jamzad M., Sadjad B.S., Mirrokni V.S., Kazemi M., Chitsaz H., Heydarnoori A., Hajiaghayi M.T., and Chiniforooshan E. A fast vision system for middle size robots in robocup. In *RoboCup-01*. Springer, Berlin. To appear in Lecture Notes in Artificial Intelligence.

[10] Lowe D. *Perceptual Organisation and Visual Recognition*. Robotics: Vision, Manipulation and Sensors. Kluwer Academic Publishers, Dordrecht, NL, 1985. ISBN 0-89838-172-X.

[11] Olson C.F. Probabilistic self-localization for mobile robots. *IEEE Transactions on Robotics and Automation*, 16(1):55–66, February 2000.

[12] Parhami B. Voting algorithms. *IEEE Transactions on Reliability*, 43(3):617–629, 1994.

[13] Thrun S., Fox D., Burgard W., and Dellaert F. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2000.

[14] The RoboCup Federation. RoboCup-99 Stockholm, Award Winners. *http://www.robocup.org/games/99stockholm/3132.html*.

[15] The RoboCup Federation. RoboCup-2000 Melbourne, Award Winners. *http://www.robocup.org/games/2000melbourne/3144.html*.